

背後の付箋を一番手前へ！

～「配列」vs「双方向リスト」データ移動アルゴリズムを解説せよ～

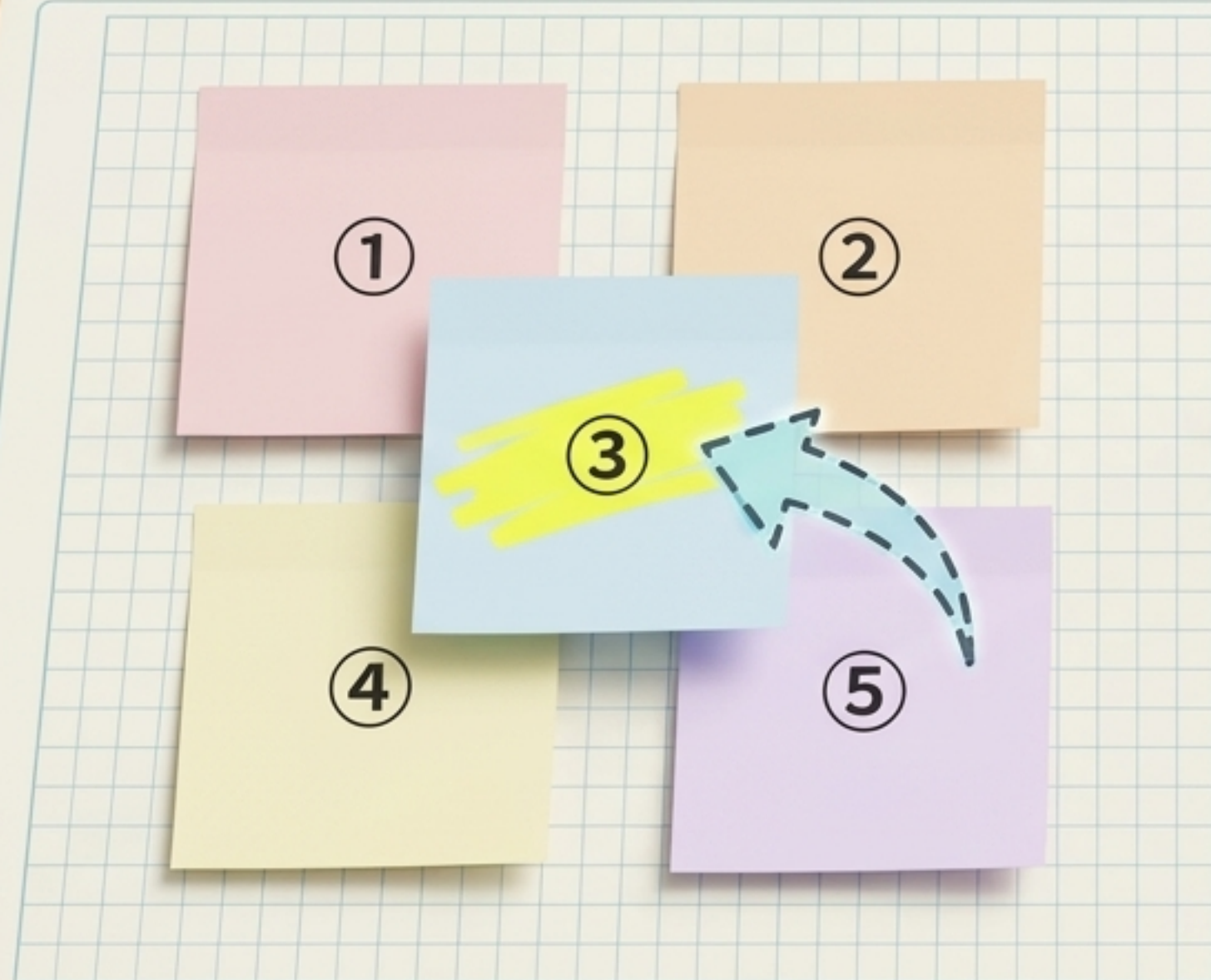
応用情報技術者試験 過去問ワークブック（解答アシスト版）



ミッション要件：指定した付箋を「末尾」へ移動させよ

画面の一番手前に表示するには、データ構造の「一番最後（末尾）」にデータを配置し直す必要がある。

(Concept)



手前（画面の最前面）

(Data Rule)



データ構造の『末尾』

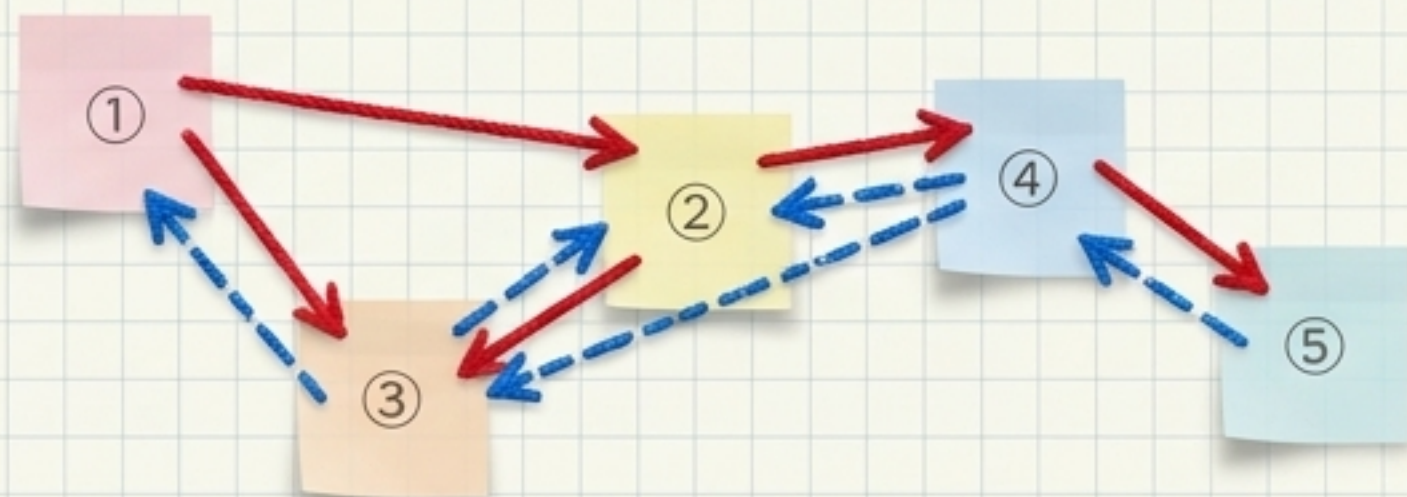
挑戦する2つのデータ構造

配列 (Array)



整然と並んだ小箱の集合。
番号（添字）で場所を管理する。

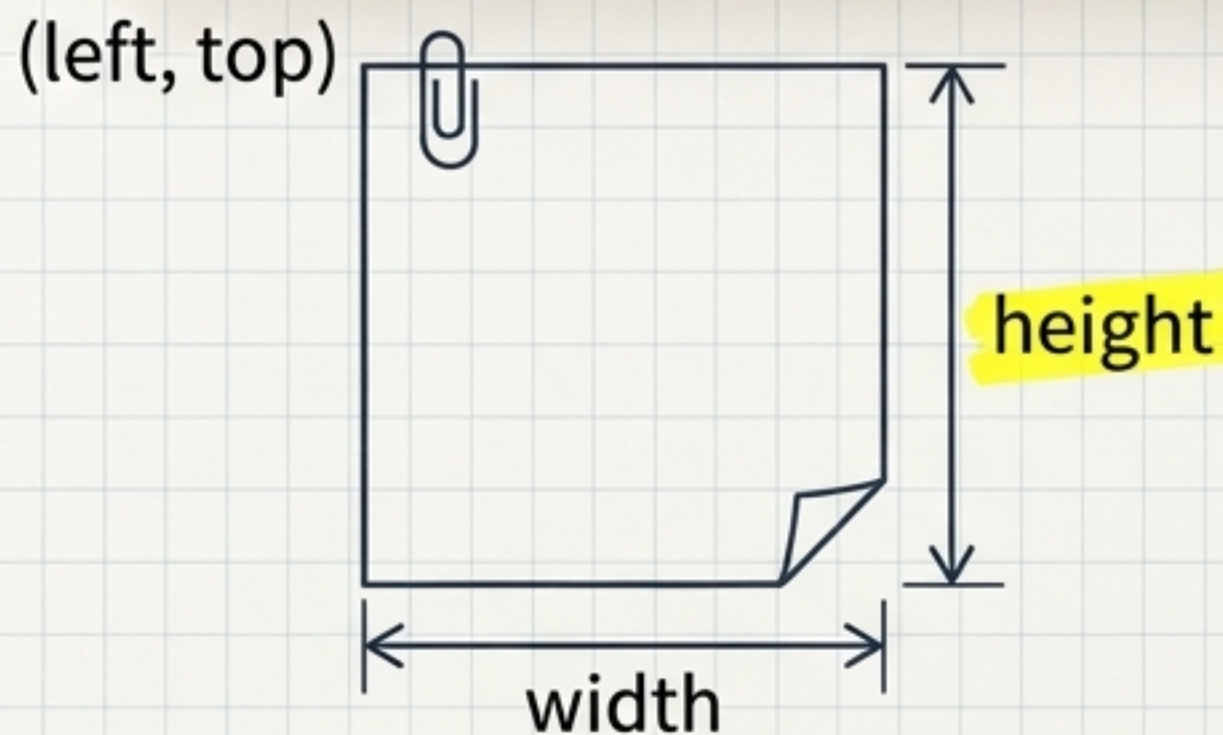
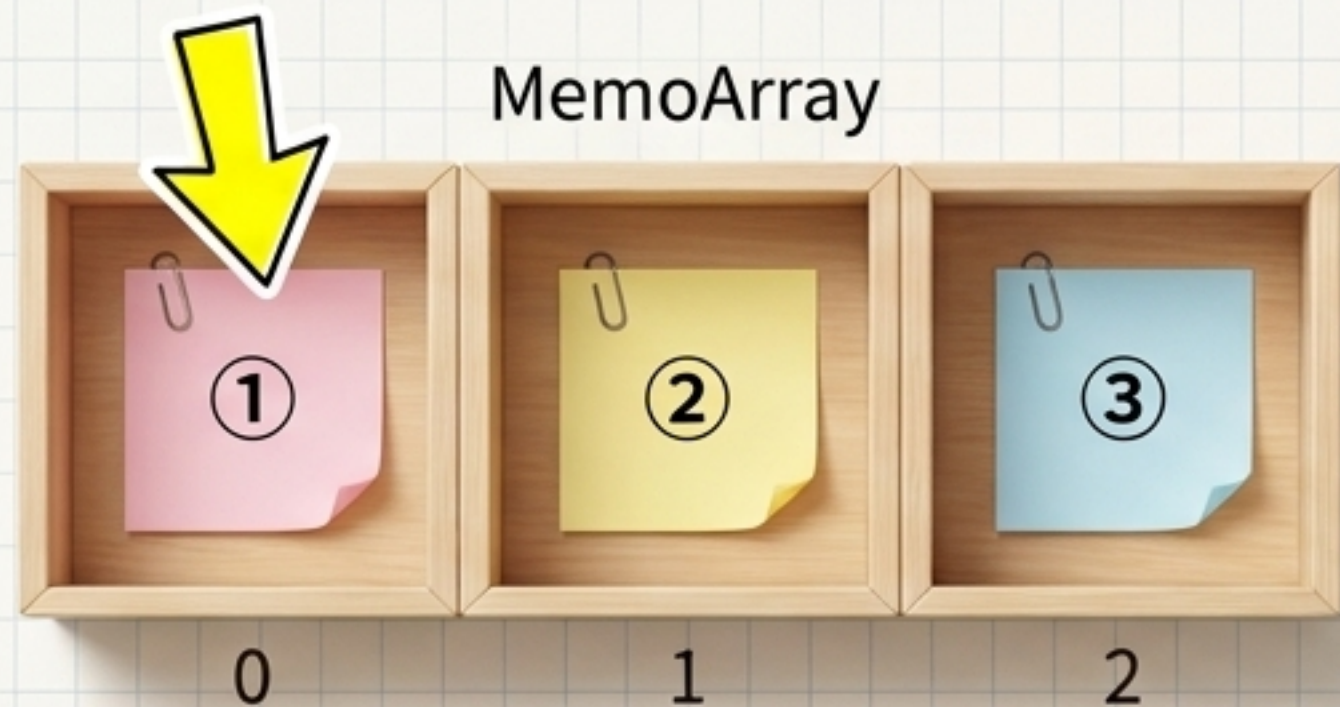
双方向リスト (Doubly Linked List)



カラフルな糸（ポインタ）で
数珠つなぎにされたデータ。

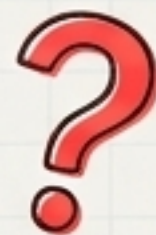
[設問1(1)のヒント] 配列データへのアクセス

Q. 配列に格納された付箋①の「高さ」を求める式は？



Step 1: 付箋①が入っている
箱の番号 (添字) は？

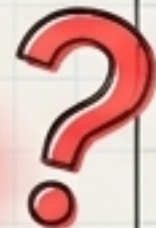
MemoArray[?]



Step 2:

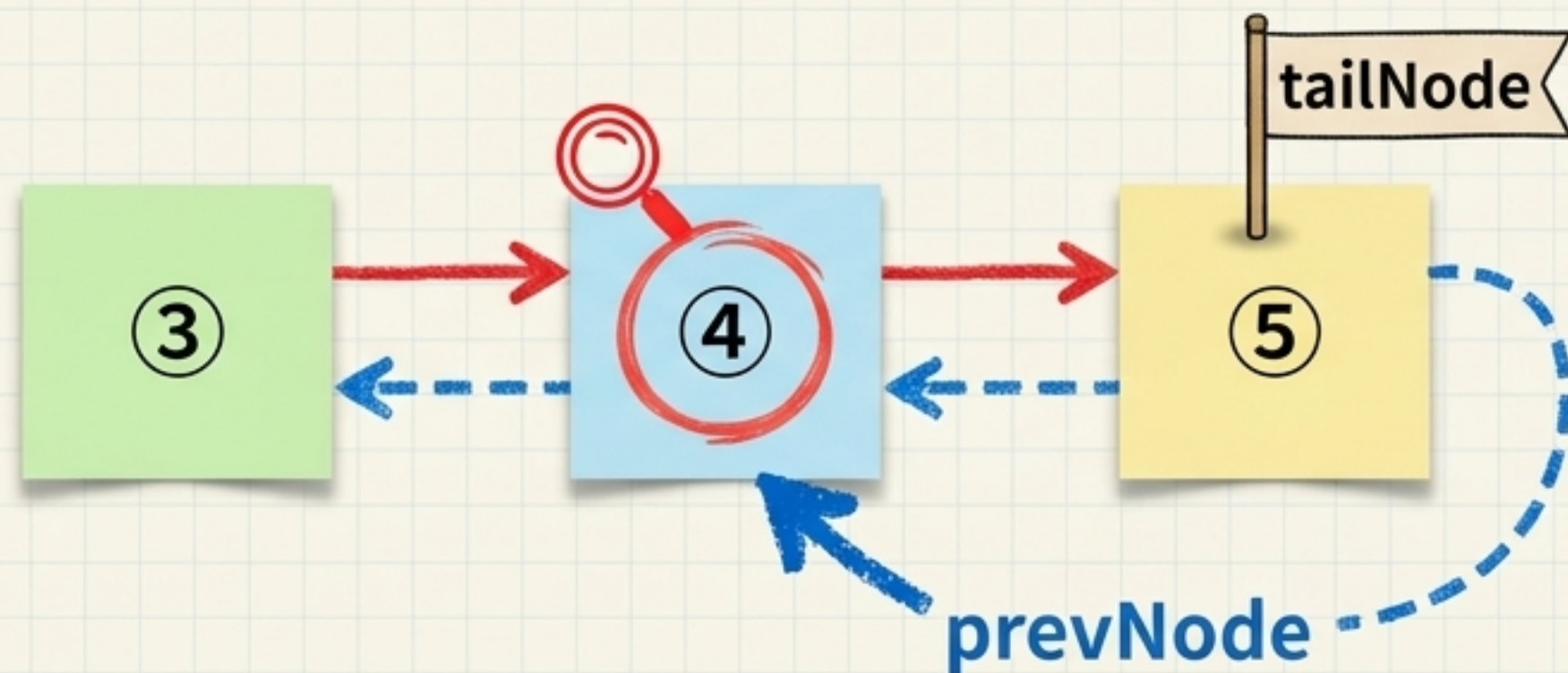
- 構造体から「高さ(height)」を
取り出すときの記号ルールは？

[配列の場所] . [要素名]

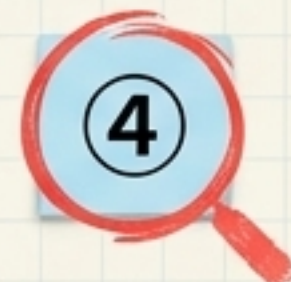


[設問1(2)のヒント] リストデータへのアクセス

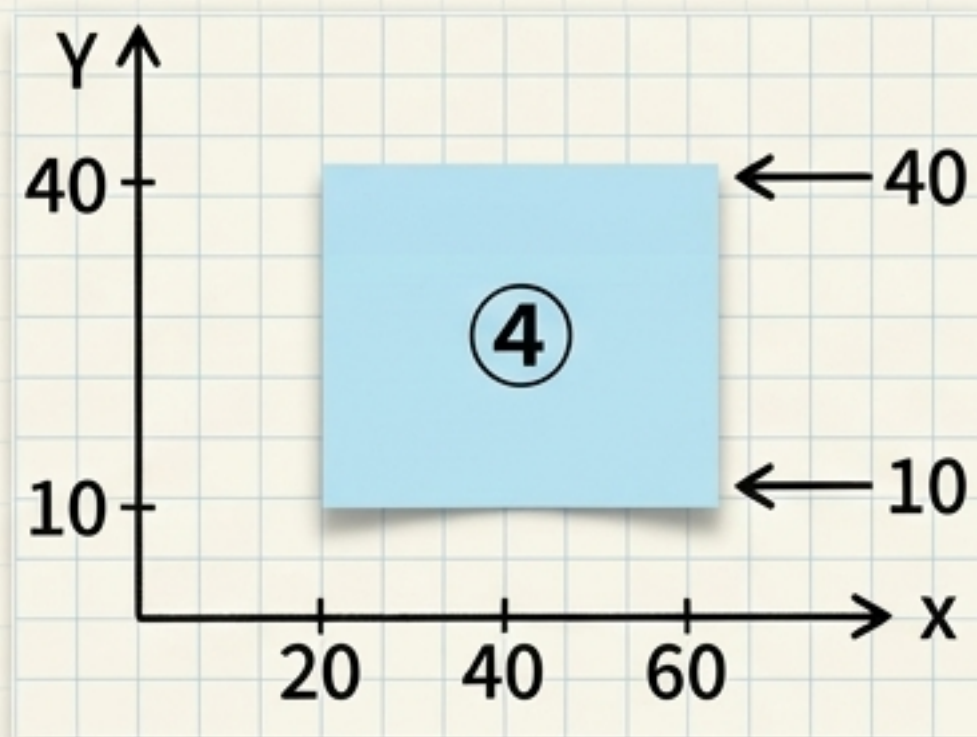
Q. `tailNode.prevNode.data.height` の値は？



Trace: `tailNode`から1つ「前(prev)」に戻ると、どの付箋に着く？



Calculate: その付箋の座標データ (図1) から、高さを計算しよう。

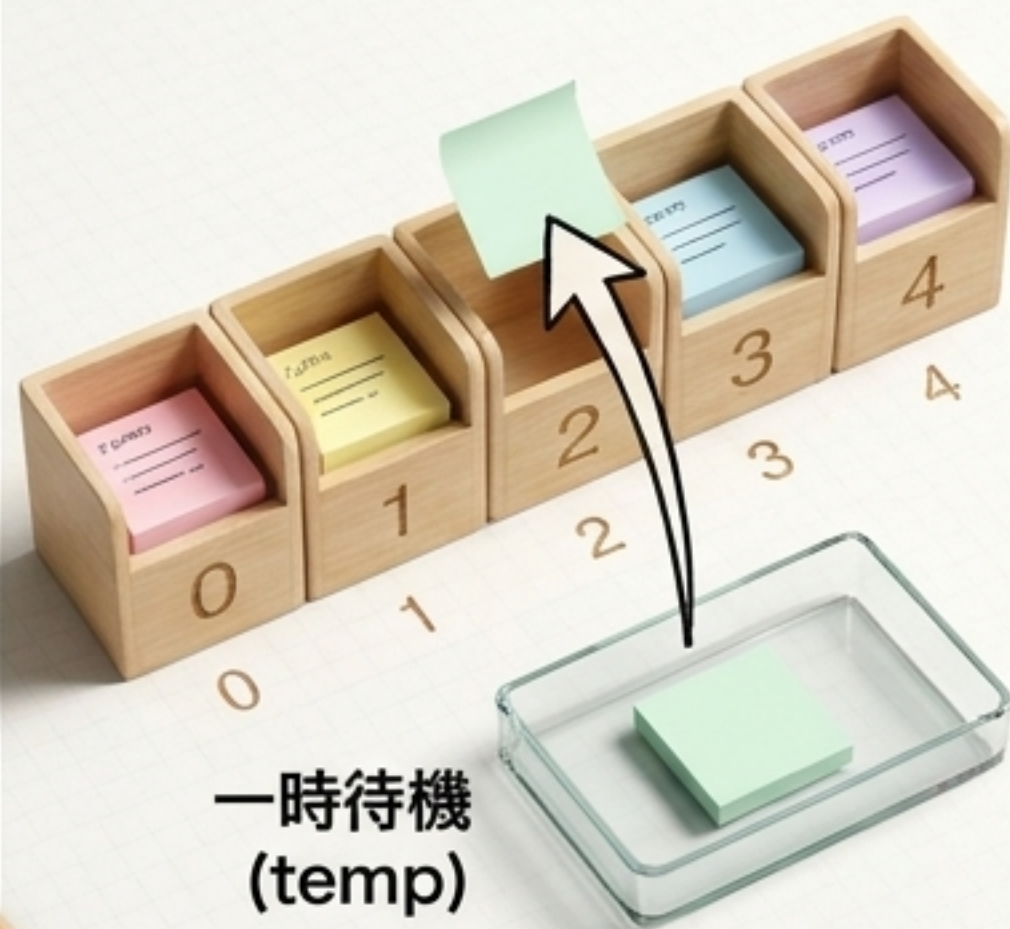


$$\left[\begin{array}{c} \text{上端の} \\ \text{座標} \end{array} \right] - \left[\begin{array}{c} \text{下端の} \\ \text{座標} \end{array} \right] = \left[? \right]$$

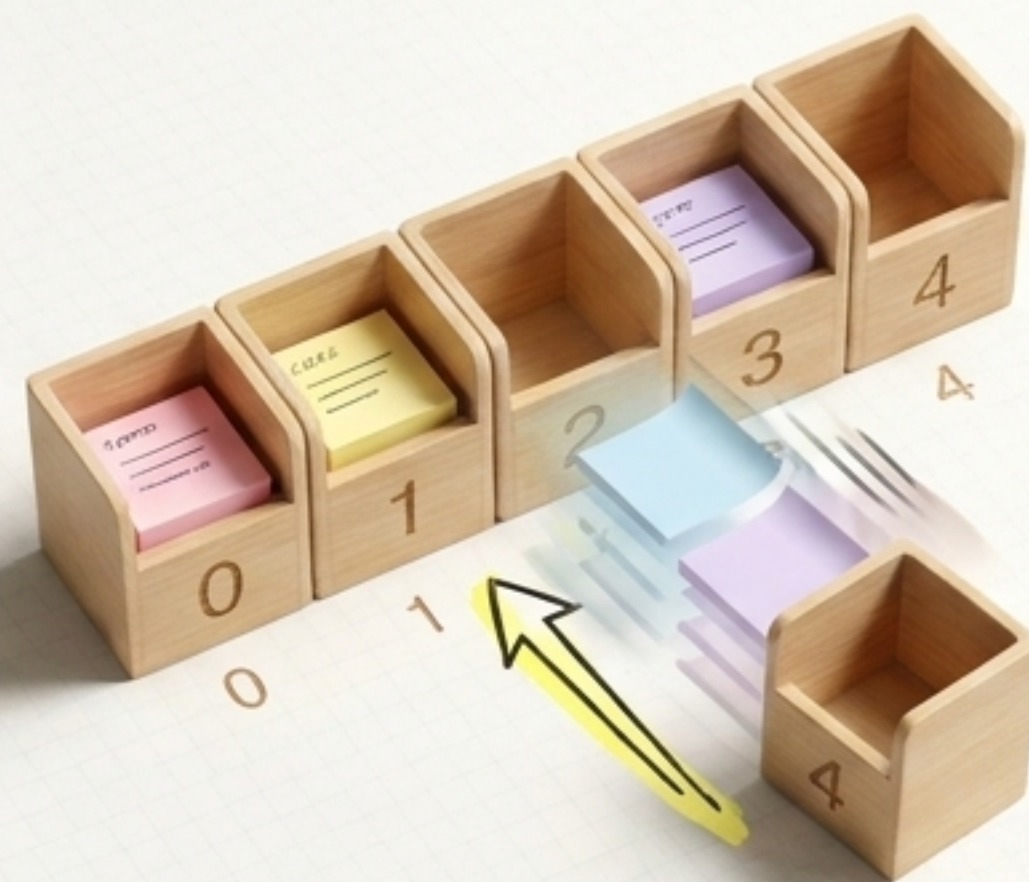
配列の移動戦略：抜いて、詰めて、最後に入れる

配列には途中に「空き部屋」を作れない。全員で協力してスペースを作る必要がある。

1. 抜く (退避)



2. 詰める (シフト)



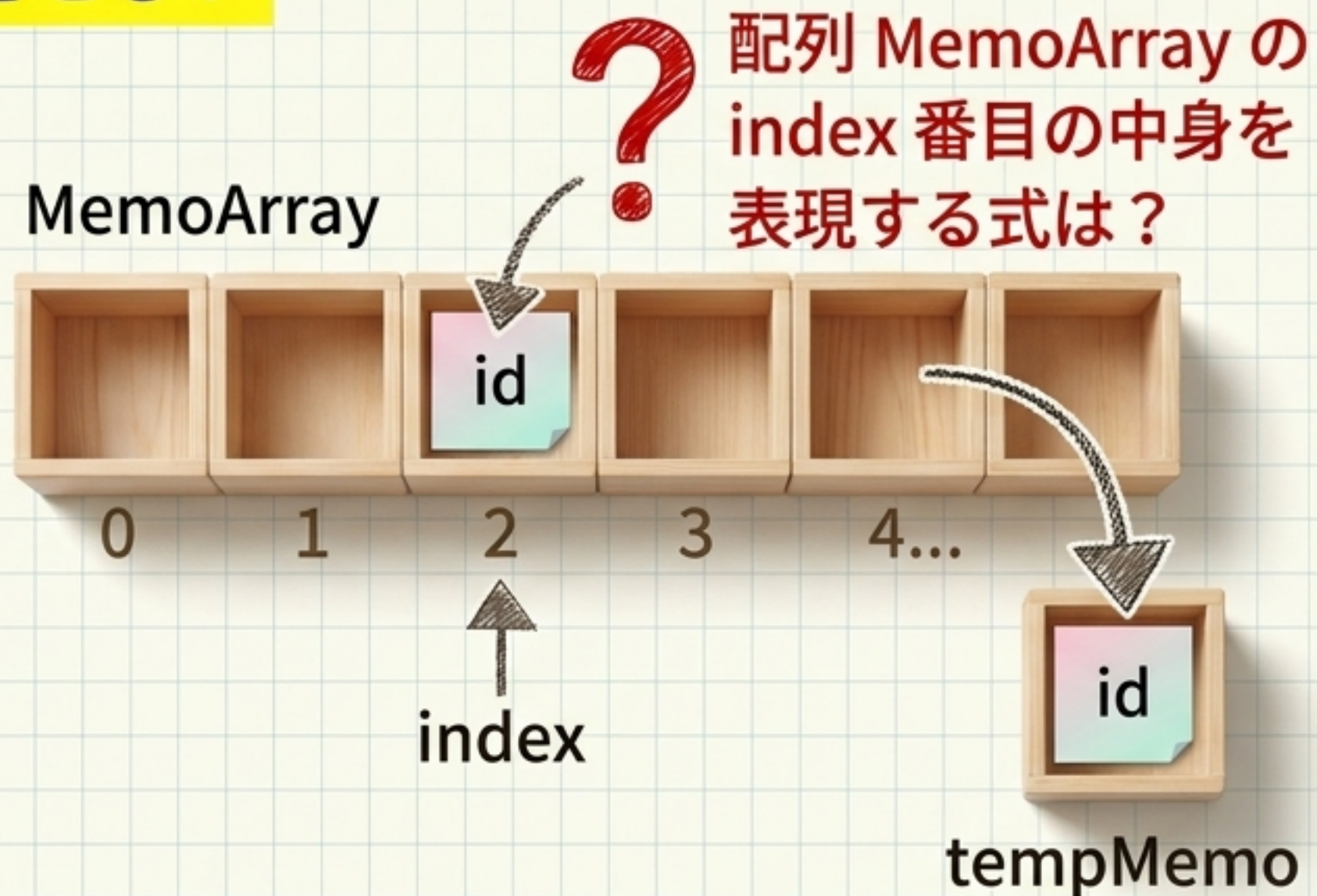
3. 入れる (代入)



[設問2 ア のヒント] ターゲットの退避

Q. 一時変数 tempMemo に何を退避させる？

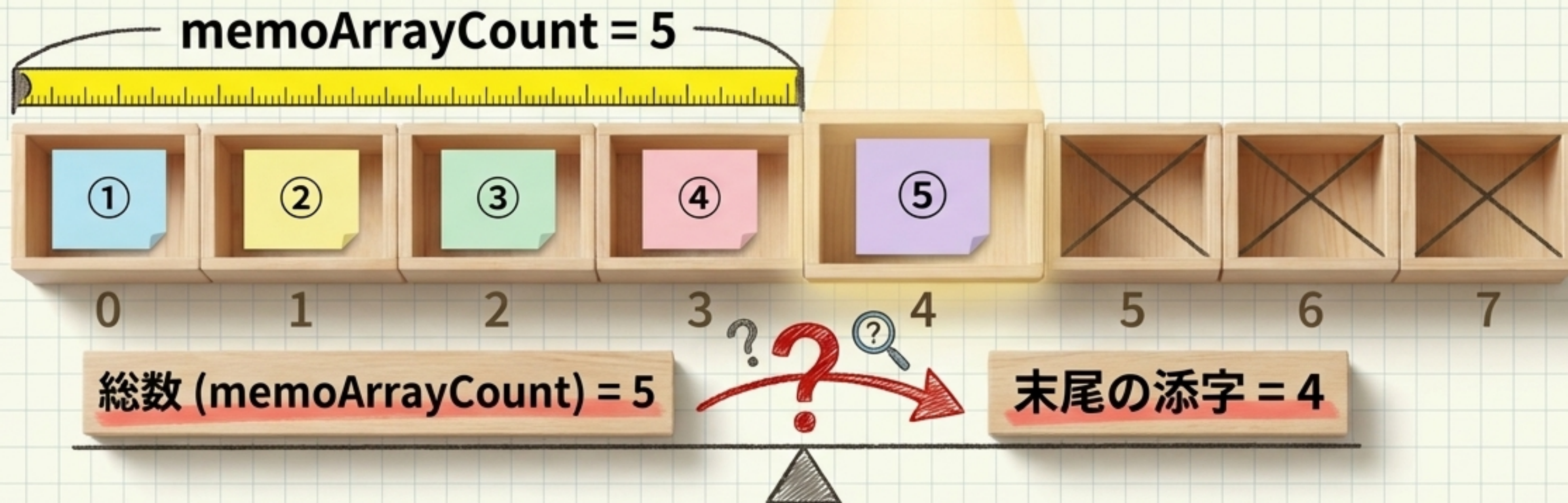
```
function moveForeArray(id){  
  index ← findArrayIndex(id)  
  tempMemo ← [ ア ]  
}
```



直前のコードで、移動させたい付箋の「場所（添字）」は変数 index に特定されている。

[設問2 イ のヒント] どこまでずらす？ 末尾はどこ？

Q. データの入っている「末尾」の場所を示す式は？

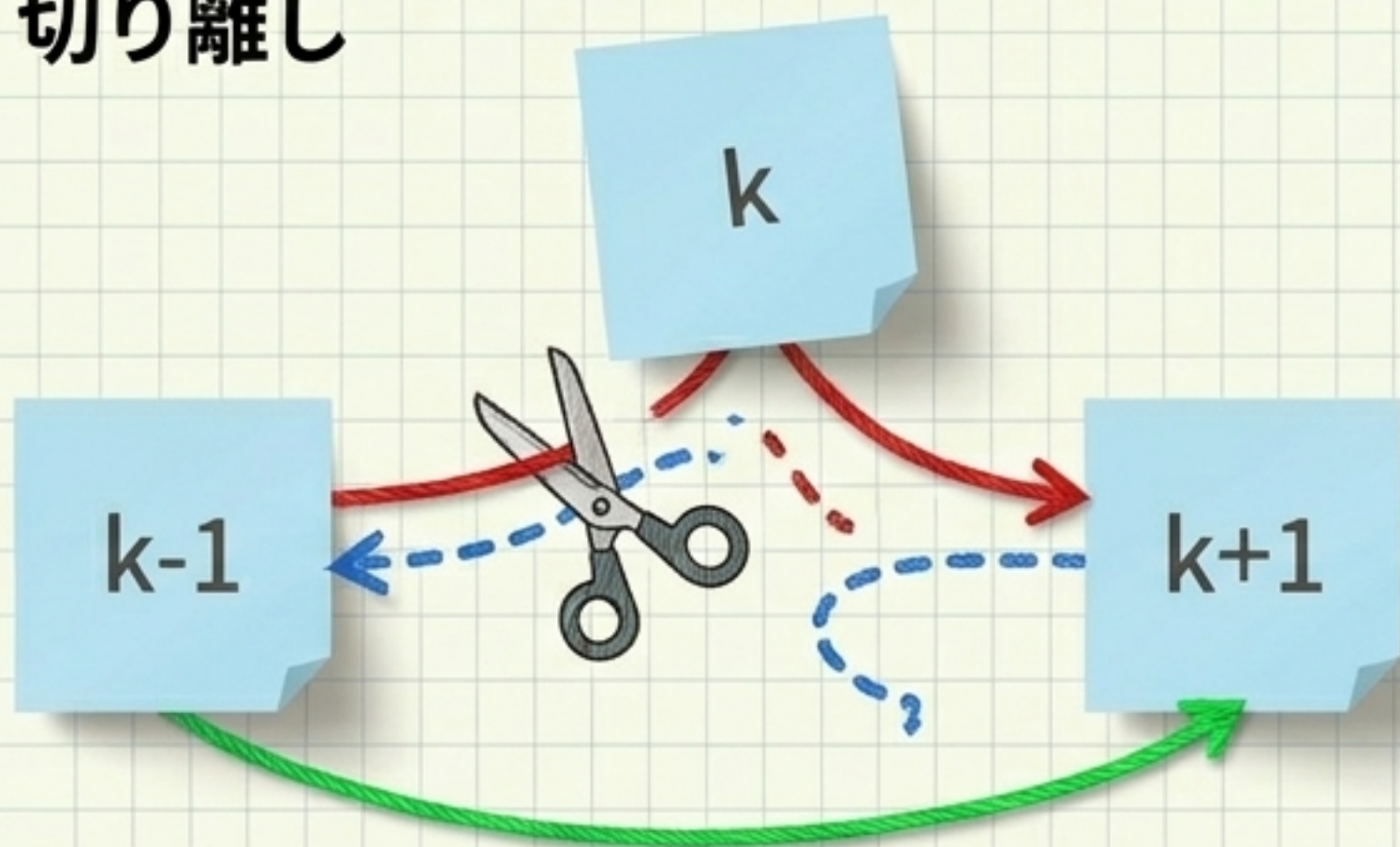


- 変数の確認：`memoArrayCount` = 現在入っているデータの総数。
- 注意点：配列の添字は「0」から始まる。
- 総数が「5」のとき、一番最後の部屋の番号は「？」になる法則性を考えよう。

リストの移動戦略：切り離して、末尾にくっつける

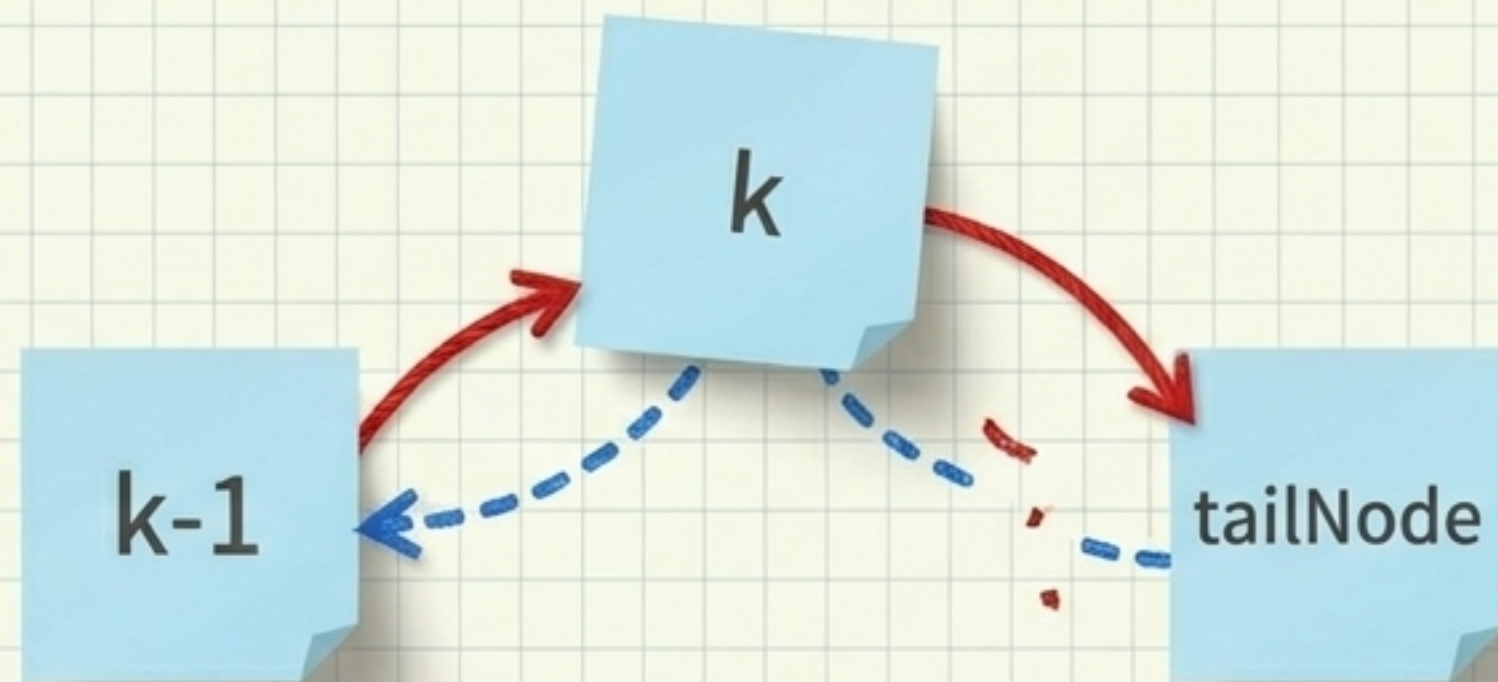
リストは全員が移動する必要はない。「特定の糸（ポインタ）」を付け替えるだけで完結する。

切り離し



- $k-1$ 細期が移動に切りを離れし。
- 「データ」を付け替る再」で完結する。

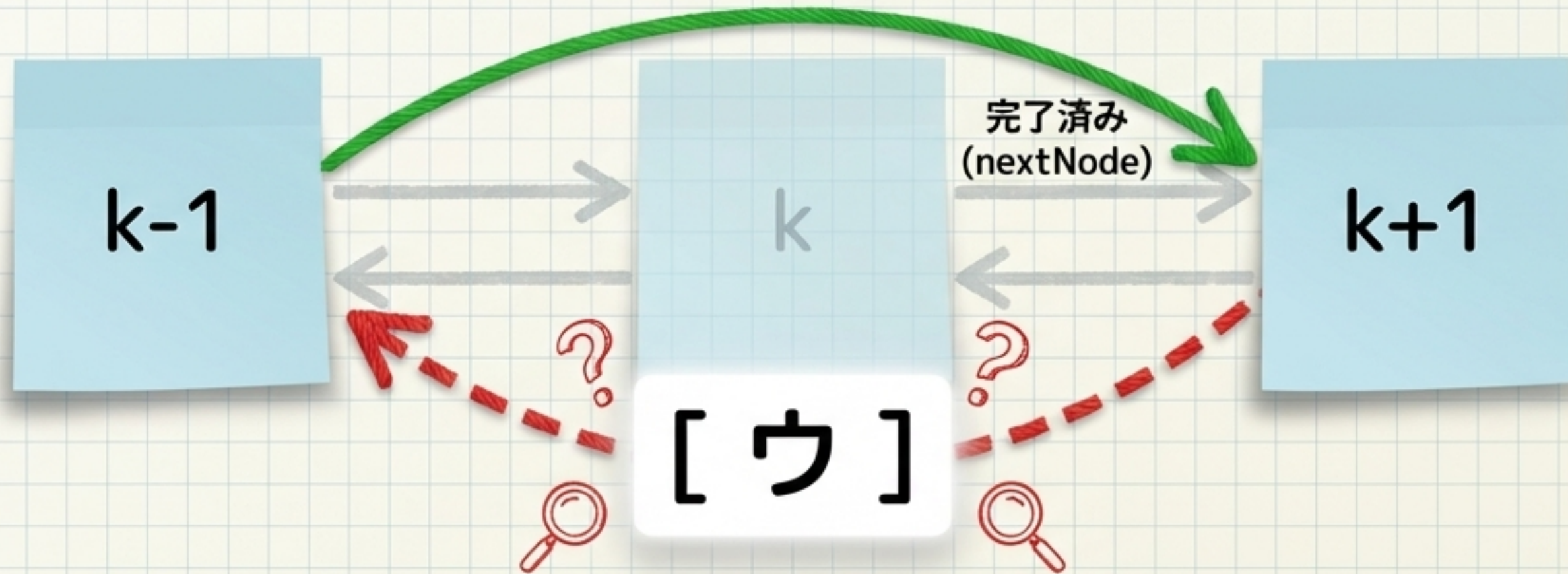
末尾へ連結



- k と 隣の連結は「末尾にくつつ連結」
- 末尾の連結が打持されていほかを筋する。

[設問3 ウ のヒント] 切り離しのポインタ操作

Q. 後ろのノード(k+1)から、前のノード(k-1)へ逆方向の橋を架けるには？



node.nextNode
(k+1に到着)

その .prevNode の向き先を...

node.prevNode (k-1)
に設定する！

- ・ 直前のコードで「k-1からk+1」への前進ルートは完成済み。
- ・ 双方向リストなので、「k+1から見た『前』」も「k-1」に更新しなければならない。
図とコードを見比べて式を組み立てよう。

[設問3 エ のヒント] 末尾への連結

Q. 移動してきたノード(k)の「前」は、誰を指すべき？



ヒント：自分自身(node)の「前(prevNode)」に、「元の末尾(tailNode)」を代入する式を作ろう。

- 「元の末尾(n)の次を、kにする」処理は完了済み。
- 残るは、移動してきた「ノードkから見た『前』」を更新する処理。これは一体誰を指すように設定すればよい？

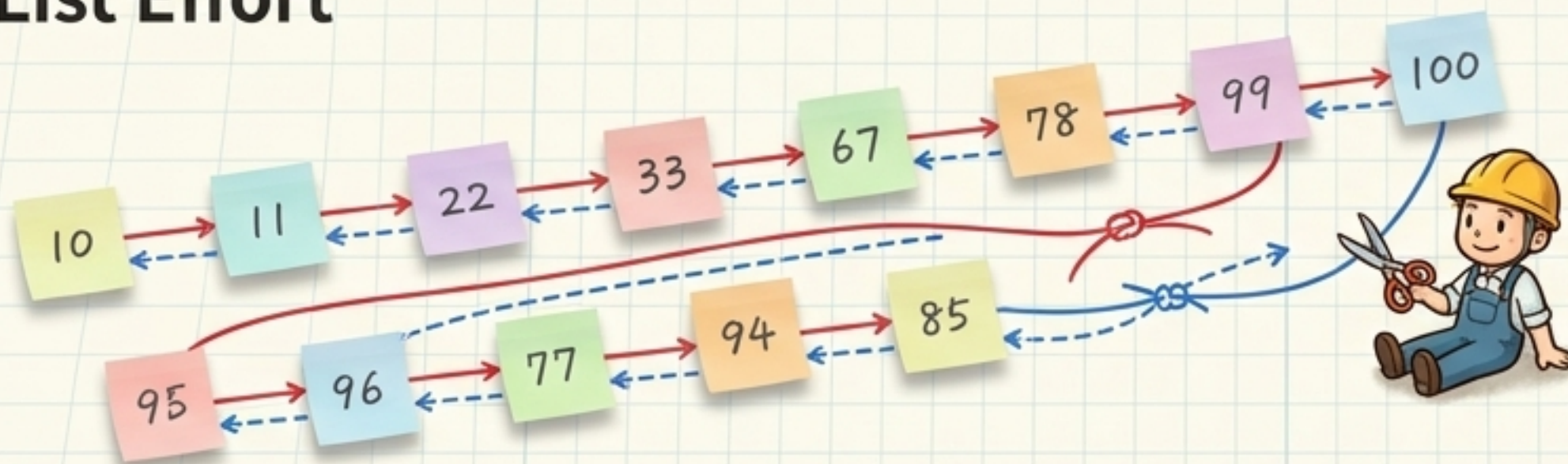
データ移動の手間（時間）はどう違う？

Array Effort



配列: ターゲットより後ろにある「すべてのデータ」をコピーして動かす大工事。

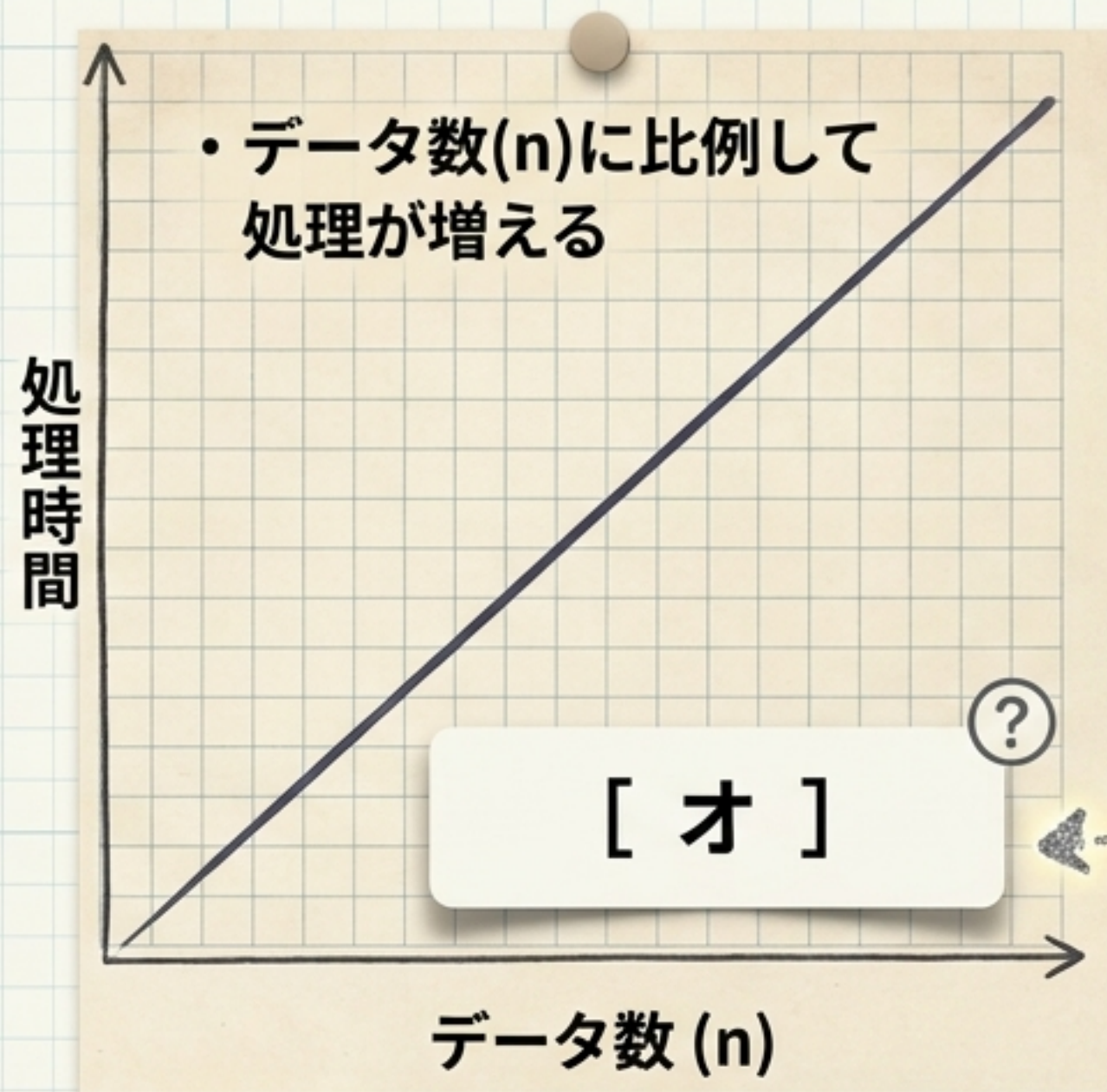
List Effort



リスト: データの総数に関係なく、ターゲット周辺の「数本の糸」を繋ぎ変えるだけのスマートな作業。

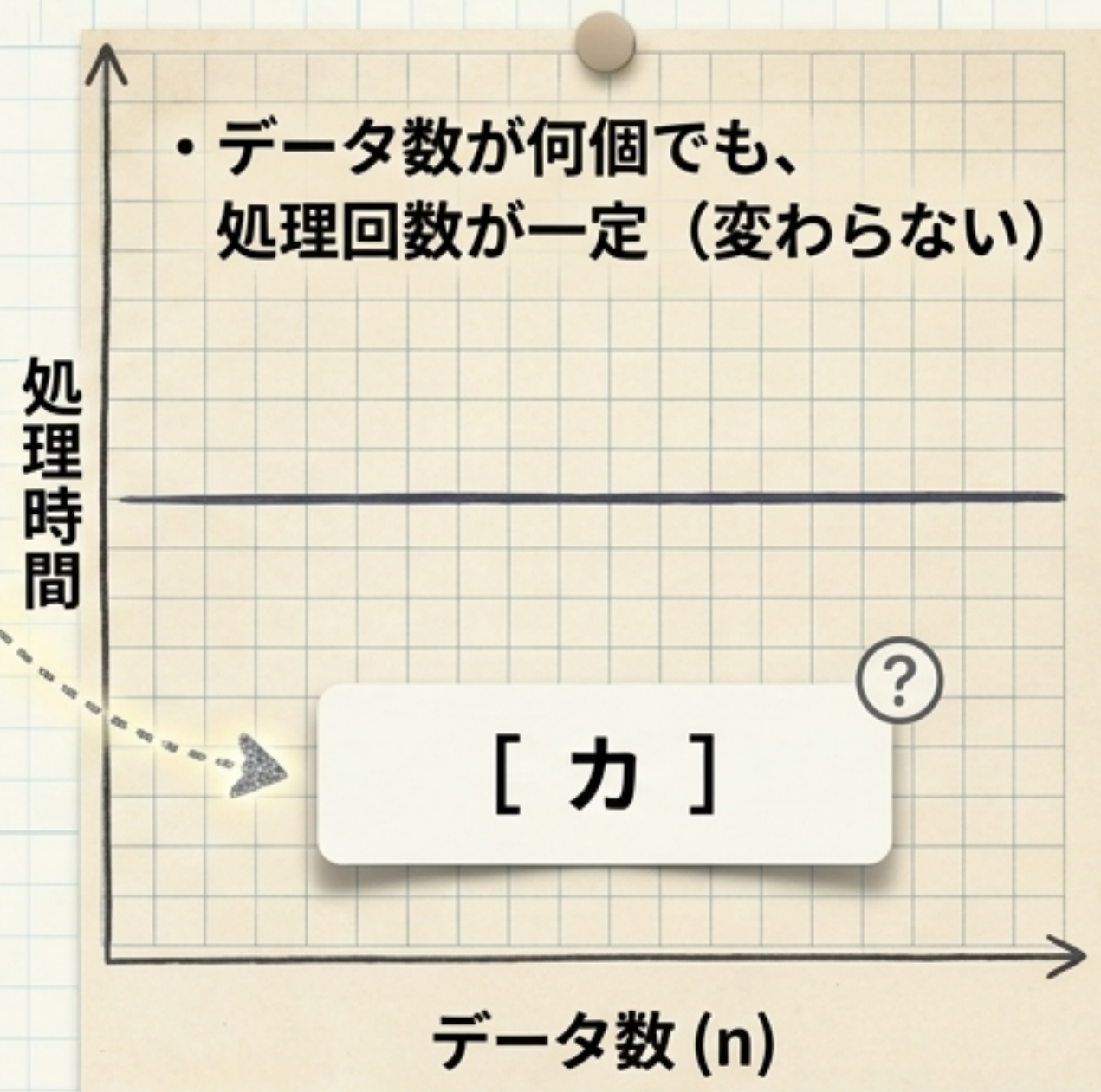
[設問4 オ・カのヒント] 計算量の違い (O記法)

Q. データ数(n)が増えたとき、処理時間はどう変化する？



$O(1)$

$O(n)$



【設問4 キ のヒント】 必要な領域（メモリ）の違い

Q. 配列は全体の領域をどう確保しなければならない？



リスト：付箋が増えるたびに、
必要な分だけ場所を用意する。

表の中にヒントが！

表 使用する構造体、配列、定数、変数及び関数		
名称	種類	内容
Memo	構造体	一つの付箋のデータ構造。次の値を管理する構造体である。 id: 付箋の一意な ID, text: 付箋のテキスト内容 x, y: 付箋の位置, width, height: 付箋の幅, 高さ
MEMO_MAX_SIZE	定数	デスアトップに配置できる付箋の最大数。
MemoArray	配列	構造体 Memo を配列として扱う。変数 memoArray は MEMO_MAX_SIZE の配列。配列の要素は、MemoArray[i] と表記する (12 配列の要素)。 配列の要素は必ずしも配列の要素とする。
memoArrayCount	変数	配列 MemoArray に格納されている付箋データの総数。
moveForArrayId	関数	付箋 ID が指定された付箋データを配列 MemoArray から本表へ移動する。



配列：最初に「巨大な棚」を用意し
たら、サイズは変えられない。
では、あらかじめ「どれくらい」
の棚を用意すべきか？

まとめ：要件によって最適なデータ構造は変わる

	配列 (Array)	リスト (List)
特定のデータの見つけやすさ	○	×
データの移動・追加の速さ	×	○
メモリ領域の無駄の少なさ	×	○

- ✓ 検索スピード重視なら「配列」。
- ✓ 移動や追加のスピード重視なら「リスト」。

すべてのピース（ヒント）は出揃った！自分の手で解答欄を埋めてみよう。